(12) UK Patent Application (19) GB (11) 2 264 796 (13) A

(43) Date of A publication 08.09.1993

(21) Application No 9204440.3

(22) Date of filing 02.03.1992

(71) Applicant
International Business Machines Corporation

(Incorporated in the USA – New York)

Armonk, New York 10504, United States of America

(72) Inventors
Susan Malaika
Michael John Brady
Geoffrey Meacock

(74) Agent and/or Address for Service
Robert Douglas Moss
IBM United Kingdom Limited,
Intellectual Property Department, Mail Point 110,
Hursley Park, Winchester, Hampshire, SO21 2JN,
United Kingdom

(51) INT CL[5]
G06F 9/46

(52) UK CL (Edition L)
G4A APX

(56) Documents cited
EP 0457117 A2

(58) Field of search
UK CL (Edition K) G4A APX AUD
INT CL[5] G06F 9/46
On-line database: WPI

(54) Distributed transaction processing

(57) A method and system for distributed transaction processing between first and second transaction processing means in communication with each other employ a client program on the first transaction processing means which calls a server program on the second transaction processing means. A first type of server program call permits only the first transaction processing means to commit resources for both client and server, both programs effectively being part of a single transaction. A second type of program call permits the second transaction processing means to commit resources independently so that the server program runs as part of a separate transaction.

GB 2 264 796

Patents Act 1977 27
Examiner's report to the Comptroller under Section 17 (The Search Report)

Application number
GB 9204440.3

Relevant Technical fields

(i) UK Cl (Edition K) G4A (APX, AUD)

G06F 9/46

(ii) Int Cl (Edition)

Search Examiner

B G WESTERN

Databases (see over)

(i) UK Patent Office

(ii) ONLINE DATABASE: WPI

Date of Search

11 SEPTEMBER 1992

Documents considered relevant following a search in respect of claims 1 TO 11

Category (see over)	Identity of document and relevant passages	Relevant to claim(s)
A	EP 0457117 A2 (IBM) NB pages 4-5	1-11

Category	Identity of document and relevant passages	Relevant to claim(s)

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family, corresponding document.

Databases: The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journal (Patents). The...

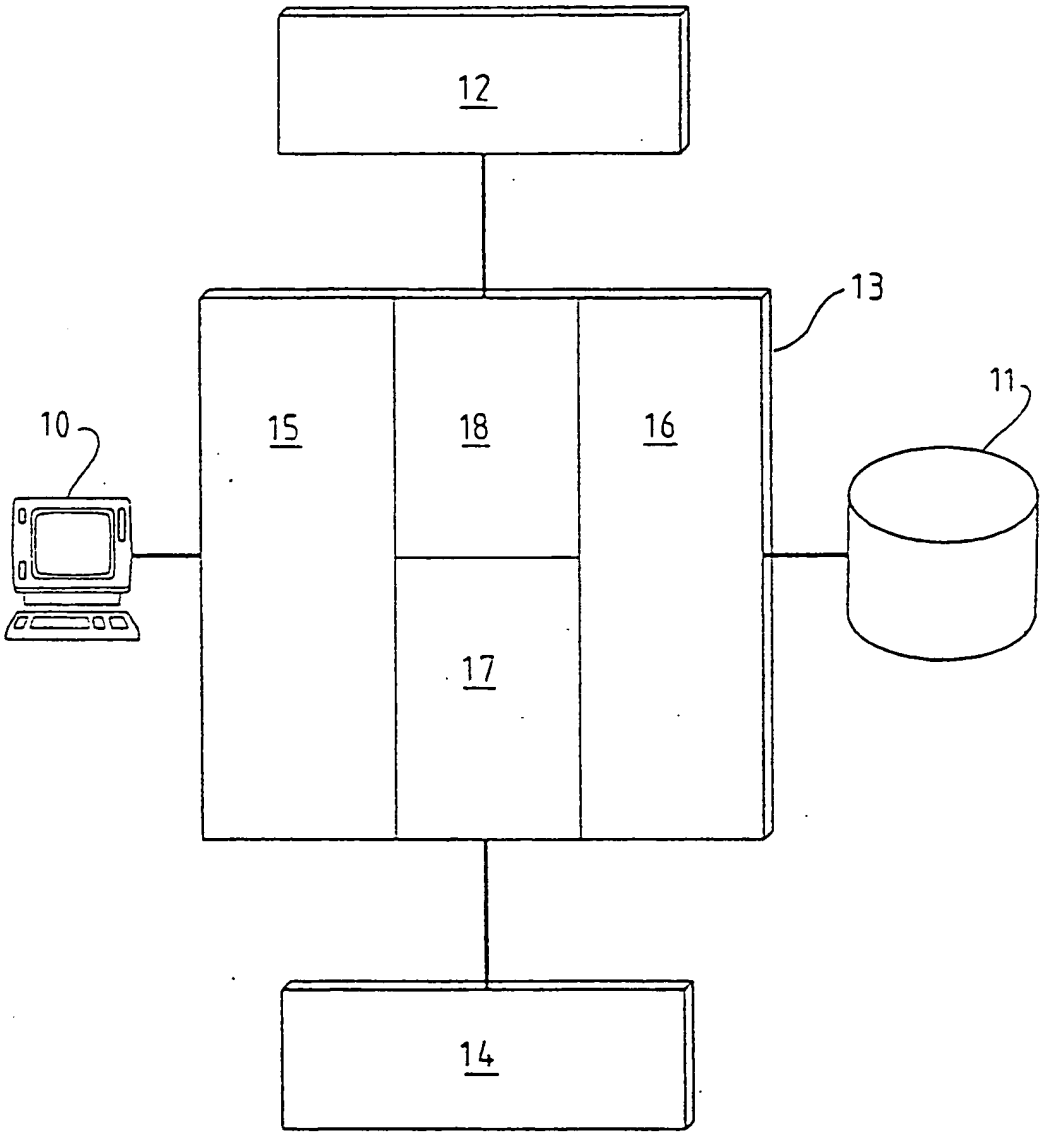


FIG. 1

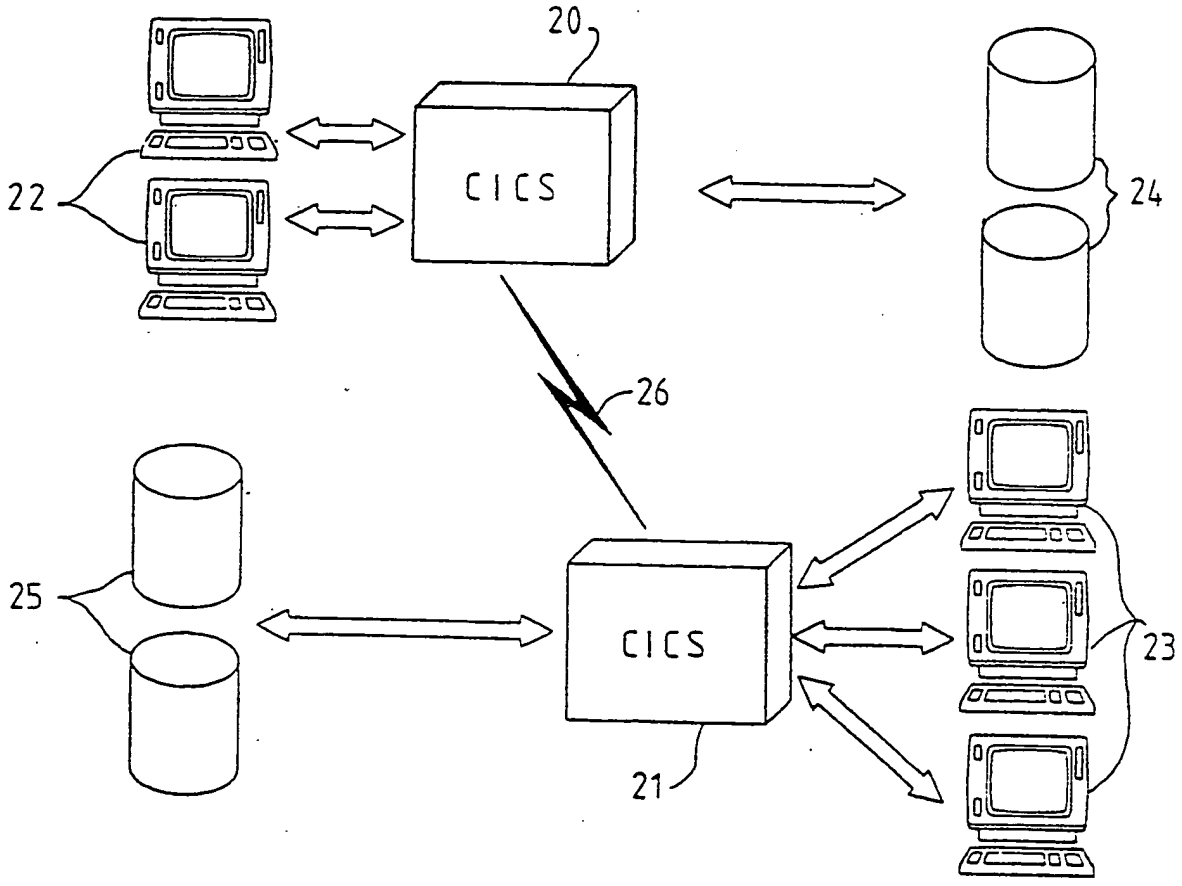


FIG. 2

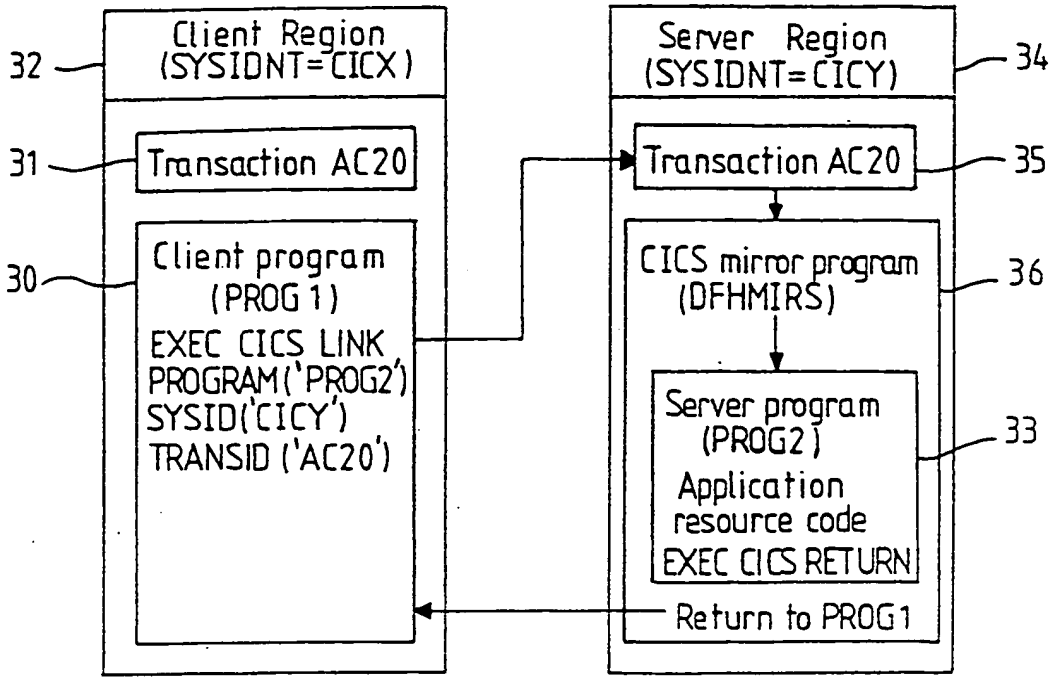

Client Region
(SYSIDNT=CICX)
32
Server Region
(SYSIDNT=CICY)
34
31
Transaction AC20
Transaction AC20
35
30
Client program
(PROG 1)
EXEC CICS LINK
PROGRAM('PROG2')
SYSID('CICY')
TRANSID ('AC20')
CICS mirror program
(DFHMIRS)
36
Server program
(PROG2)
Application
resource code
EXEC CICS RETURN
33
Return to PROG1

FIG. 3

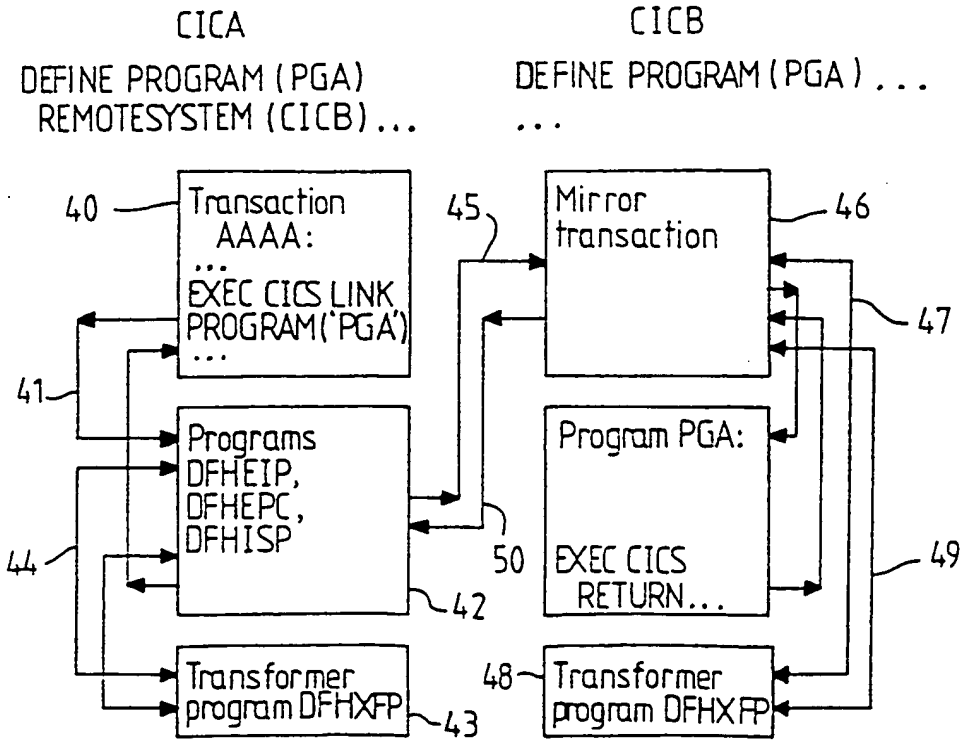

CICA
DEFINE PROGRAM (PGA)
REMOTESYSTEM (CICB)...
CICB
DEFINE PROGRAM (PGA)...
...
40
Transaction
AAAA:
...
EXEC CICS LINK
PROGRAM('PGA')
...
45
Mirror
transaction
46
47
41
Programs
DFHEIP,
DFHEPC,
DFHISP
Program PGA:
EXEC CICS
RETURN...
44
50
49
42
Transformer
program DFHXFP
43
48
Transformer
program DFHXFP

FIG. 4

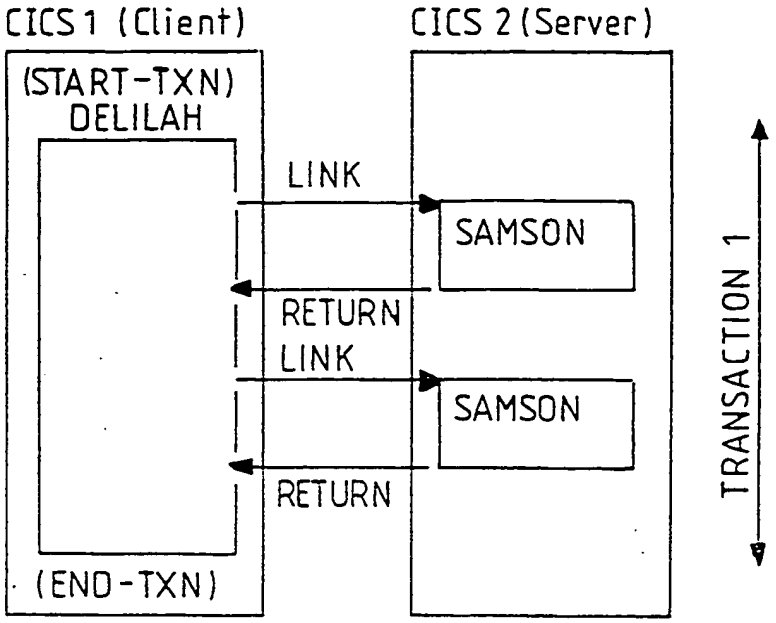

CICS 1 (Client)
CICS 2 (Server)
(START-TXN)
DELILAH
LINK
SAMSON
RETURN
LINK
SAMSON
RETURN
(END-TXN)
TRANSACTION 1

FIG. 5

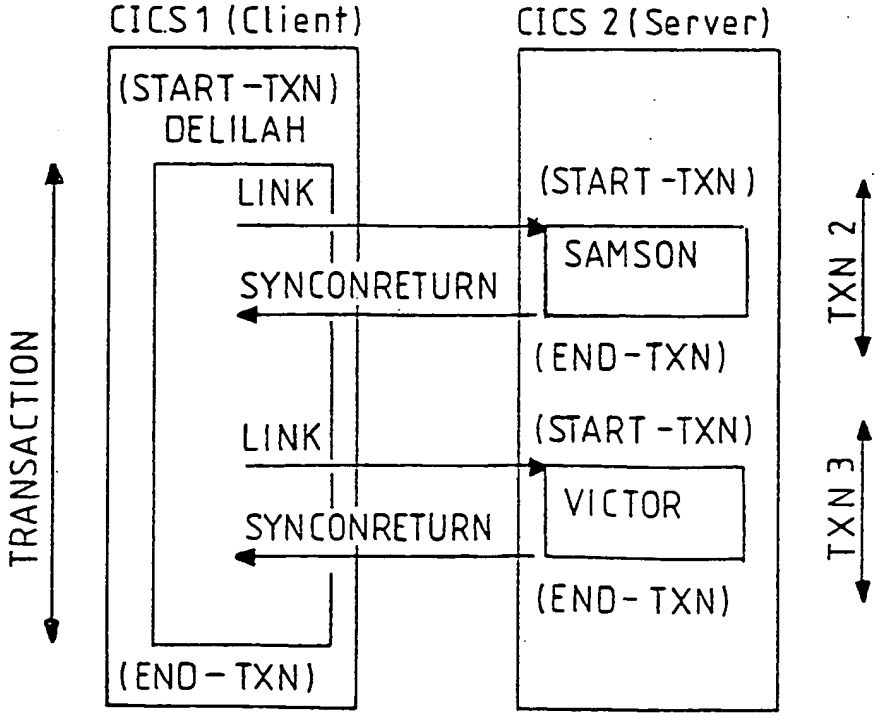

CICS 1 (Client)
CICS 2 (Server)
(START-TXN)
DELILAH
LINK
(START-TXN)
SAMSON
SYNCONRETURN
(END-TXN)
LINK
(START-TXN)
VICTOR
SYNCONRETURN
(END-TXN)
(END-TXN)
TRANSACTION
TXN 2
TXN 3

FIG. 6

DISTRIBUTED TRANSACTION PROCESSING

Field of the Invention

The present invention relates to a method of and a system for distributed transaction processing.

Background of the Invention

On-line transaction processing (OLTP) relates to the real time processing of business and commercial transactions such as room or seat reservation and financial transactions. An OLTP system typically supports a network of thousands of terminals and provides near instant access to data held in data sets or databases. To do this it must support the data-communication links between terminals, processors and data.

One very well known family of OLTPs is the IBM Customer Information Control System (CICS) which is described in the publication CICS General Information (GC33-0155-4), 5th Edition, October 1990, from IBM Corporation. CICS is an application enabling program which runs on a general purpose operating system and provides a common transaction processing support environment to customer application programs via an Application Program Interface (API).

A transaction, in more detail, is a piece of processing initiated by a single request usually from a terminal. A single transaction consists of one or more application programs, that when run, will carry out the processing needed. A transaction is also referred to as a logical unit of work (LUW) in CICS and is also said to be "atomic" in the sense that it may only fail or complete its task, but not partly complete it.

A transaction operates on resources (or entities) which, in the case of CICS, can be keyed and direct access files, sequential files and queues. Transactions can only have one of two outcomes: "committed" or "aborted". If aborted, certain "protected" actions must be redone or undone if irretrievable damage to customer data or propagation of errors is to be avoided.

In either event, the transaction and the recoverable resources it was acting on must be backed up to the state they were in before the transaction started. Transactions are delimited in practice by commit points (or SYNCPOINTS) at which changes to resources are committed so that the resources are restored to their state at the previous SYNCPOINT.

In order to achieve this, it is normal practice in CICS and other OLTP systems to provide Redo and Undo logs of the actions taken since the last SYNCPOINT and also to lock all or selected portions of resources for the duration of a transaction. Further background on transaction processing can be found in a paper by J N Gray ("A Transaction Model", Technical Report RJ2895, IBM Corp. 1980).

Another aspect of known transaction processing systems such as CICS, is the distribution and sharing of function, resources and processing between physically separate systems on different levels, which requires intersystem communication (ISC), or between different CICS regions on the same level, known as multi-region operation (MRO).

Briefly, there are several mechanisms for such intercommunication in CICS:

"Transaction routing" enables a terminal in one CICS system to run a transaction in another CICS system;

"Function shipping" enables an application program to access resources in another CICS system;

"Asynchronous processing" enables a CICS transaction to initiate a transaction in a remote system and to pass data to it;

"Distributed transaction processing" (DTP) enables a CICS transaction to communicate with a transaction in another CICS system; and

"Distributed Program Link" (DPL) is a type of function shipping which enables an application program (the client) on one CICS region or system to link to another application program (the server) running in another CICS region or system.

The Distributed Program Link was introduced in the CICS OS/2 system and relied on the use of Advanced Program-to-Program Communication (APPC) protocols, which require relatively complicated programming to establish (Ref. CICS OS/2 System and Application Guide - SC33-0616, IBM Corp. 1991).

CICS may also communicate with external shared databases and the CICS/ESA 3.2 system handled such communication through a Resource Manager Interface (RMI). Great care is needed to ensure that the integrity of shared data is preserved and, for that reason, a so-called two-phase commit protocol is employed to prevent inconsistent and uncontrolled changes of the database by multiple data or transaction processing systems. (Ref. CICS Task Related User Exits: An implementation for co-ordinating database commits, GG22-0497, IBM Corp. 1991). A certain amount of time is required for completion of the two-phase commit protocol and this, to some extent, reduces the performance of the system.

Proposals have also been made in the prior art for various forms of interrelated transactions which may be useful in a distributed computing environment. So-called nested transactions are described in "Nested Transactions - An Approach to Reliable Distributed Computing" (J E B Moss, MIT Press, 1985). Both portions of a nested transaction in fact run as a single unit of work and incur performance overheads.

The book "Camelot and Avalon - A Distributed Transaction Facility" (Ed Eppinger et al, Morgan Kaufmann Publishers Inc. 1991) also describes nested top-level transactions running on the same system, as opposed to remotely. The same book also describes wrapped server calls in which servers are called by programs that do not run within the scope of a transactions manager. Such server programs cannot issue intermediate commit messages.

Disclosure of the Invention

The prior art has hitherto provided relatively inflexible distributed transaction processing incurring significant overheads of additional programming or performance.

According to the present invention, there is provided a method of processing transactions in a distributed transaction processing system comprising first and second transaction processing means capable of communicating with each other and with resources which may be used and updated in the processing of transactions, each of the transaction processing means being adapted to issue a commit message indicating, at the end of a respective transaction, that changes to said resources shall be committed; the method comprising the steps of initiating a transaction on said first transaction processing means, said transaction including a client program resident in said first transaction programming means which includes a program call to a related server program resident in said second transaction processing means; passing said program call to said server program to cause said server program to run in said second transaction processing means; in response to a first type of program call, permitting only the first transaction processing means to issue a commit message in accordance with a commit protocol so that both client and server programs run as part of a single transaction and any resource updates by either program are only committed in response to the commit message from the first transaction processing means; and in response to a second type of program call, permitting the second transaction processing means to issue a commit message

independently of the first transaction processing means so that the server program runs as part of a separate transaction and any resource updates by the server program are committed independently.

The invention also provides a distributed transaction processing system comprising first and second transaction processing means capable of communicating with each other; resources which may be used and updated by the processing means in the processing of transactions, each of the transaction processing means being adapted to issue a commit message indicating at the end of a respective transaction, that changes to said resources shall be committed; the first transaction processing means including means for executing as part of a transaction a client program which includes calls to a related server program and the second transaction processing means including means for executing such a server program; the system further including means for initiating a transaction including such a client program; means for passing a program call from such a client program to such a related server program; means responsive to a first type of program call to permit only the first transaction processing means to issue a commit message in accordance with a commit protocol so that both client and server programs run as part of a single transaction and any resource updates by either program are only committed in response to the commit message from the first transaction processing means; and means responsive to a second type of program call to permit the second transaction processing means to issue a commit message independently of the first transaction processing means so that the server program runs as part of a separate transaction and any resource updates by the server program are committed independently.

Transactions of the type in which the server program runs as part of a separate transaction are termed "wrapped transactions". A wrapped transaction is initiated and completed within another transaction that is already in-flight. The in-flight transaction is referred to as the outer transaction. Some characteristics of a wrapped transaction are as follows:

- Its recoverable resources are committed independently of the outer transaction's resources.

- All locks it acquires are freed before returning control to the calling transaction.

- It can run at the same location as the outer transaction or at a different one.

- It can issue intermediate commit points independently of the outer transaction. These do not affect the outer transaction.

- It can initiate wrapped transactions and thus act as an outer transaction with respect to other wrapped transactions.

- It can fail independently of the outer transaction.

Some characteristics of an outer transaction that initiates wrapped transactions are as follows:

- If it fails after its wrapped transactions commit, the wrapped transaction's recoverable resources remain committed and are not rolled back.

- It can initiate wrapped transactions serially or concurrently.

- It can define new transaction context characteristics for the wrapped transaction that are different to its own.

Both outer and wrapped transactions can initiate or participate in, other types of transactions, e.g. nested transactions. Thus, a distributed application may be made up of a variety of elements, some completely co-ordinated, and some more independent.

Prior art distributed applications have a significant performance overhead compared to local applications, particularly when a two (or three) phase commit protocol is used. Wrapped transactions significantly improve the performance of distributed applications where resource access and update co-ordination between remote portions of the application is not essential.

The major advantage of wrapped transactions in the method and system of the invention is improved performance in a distributed system. For example, the following savings would be made over regular distribute transactions using a full two-phase-commit protocol:

- The two flows for two-phase-commit between the outer and wrapped transaction are eliminated.
- Participant log record forcing for two-phase-commit in the wrapped transaction, with respect to the outer transaction, is eliminated.
- Any communication connection between the outer and wrapped transaction can be freed as soon as the wrapped transaction completes, instead of being held until the outer transaction commits.

In performance measurements, the path-length between two remote CICS systems (using the CICS Intersystem Communication Method) to initiate, commit and terminate a wrapped transaction was found to be roughly 70% of that needed for a regular transaction that uses two-phase-commit to co-ordinate its updates with the outer transaction.

Additionally, there are throughput benefits that arise from freeing communication connections early. Also, more concurrency can be gained by running the wrapped transactions in parallel. In addition, the wrapped transaction cannot be in an in-doubt state with respect to the outer transaction. This avoids holding long in-doubt locks on shared data after failures.

The invention will now be described, by way of example only, with reference to the following drawings in which:

Figure 1 is an overview of an online transaction processing system suitable for the implementation of distributed transaction processing according to the invention;

Figure 2 shows an overview of a distributed transaction processing system in which two of the systems of Figure 1 communicate with each other;

Figure 3 illustrates the operation of a distributed program link feature employed in the system of Figure 2;

Figure 4 shows further detail of the distributed program link of Figure 3.

Figure 5 illustrates message flow between client and server as part of a single transaction; and

Figure 6 illustrates message flow between client and server as part of multiple wrapped transactions.

Detailed Description of the Invention

Figure 1 shows a CICS transaction processing sytem including associated hardware and software. The hardware includes terminals such as 10 and databases and files such as 11.

A host computer operating system 12, such as MVS/ESA, VSE/ESA or OS/2 EE, supplies general data processing services to CICS software 13. The CICS software may be regarded as a subsidiary operating system which provides specialised on-line services to provide an environment for execution of on-line application programs 14, typically written by a customer for a specific on-line transaction processing application.

Application programs give users online access to their data and the ability to send messages to other CICS users. In addition, application programs running under CICS can communicate with other programs running elsewhere in the same computer system, or (given the appropriate network support) with other computing systems.

The CICS software 13 includes Data communication functions 15 which provide an interface between CICS and local or remote terminals to make the input or output of data easier. They provide a degree of device independence and format independence for application programs. There are also multiregion operation (MRO) and intersystem communication (ISC) facilities. Data handling functions 16 provide an interface between CICS and stored data. They allow the data to be read or updated, while preventing unauthorised access and protecting the data from corruption. CICS has interfaces to database products and to standard file access methods. CICS also has routines to handle queues and scratchpad data used within itself. Application program services 17 provide an interface between CICS and the application programs 14. System services 18 provide an interface between CICS and the operating system. They include functions to control CICS, and to share resources.

As was explained in connection with the prior art, CICS software 13 has intercommunication facilities which allow two or more separate physical systems or two or more regions within the same host to communicate and/or share terminals and other resources. The two modes of intercommunication are multiregion operation (MRO) and intersystem communication (ISC). These assist with the implementation of co-operative and distributed processing. There are several methods of intercommunication comprising function request shipping, distributed transaction processing, asynchronous processing, transaction routing and distributed program link.

The present invention is implemented in an intersystem communication configuration as illustrated in Figure 2, though it may also be applied in a multiregion operation environment. The system of

Figure 2 comprises two CICS software systems 20 and 21, each with their associated terminals 22, 23 and data files 24, 25 connected by a communications link 26.

Specifically, the invention is implemented in the system of Figure 2, by means of the Distributed Program Link (DPL) method of intercommunication, illustrated in Figure 3.

The CICS distributed program link (DPL) enables CICS application programs to run programs residing in other CICS regions by shipping program-control LINK requests. An application can be written without regard for the location of the requested programs; it simply uses program-control LINK commands in the usual way. Entries in the CICS program definition tables allow the system programmer to specify that the named program is not in the local region (known as the client region) but in a remot region (known as the server region).

Client programs can run in a CICS intercommunication environment and use DPL without being aware of the location of the server program. The location of the server program is specified in the installed program resource definition.

In Figure 3, a client program 30 (PROG1) is running as part of a transaction 31 (AC20) in a first transaction processing system 32 (CICX). The program 30 issues a program-control LINK command for a server program 33 (PROG2) running in a second transaction processing system 34 (CICY). It does this by specifying the following sequence of instructions:-

```
EXEC CICS LINK
PROGRAM ('PROG2')
SYSID ('CICY')
TRANSID ('AC20')
```

These identify the remote program name, the remote system and the name of the transaction.

The transaction AC20 is duplicated at 35 in remote system 34 and the required server program is run under control of a mirror program 36. The mirror program recreates the original LINK request and issues it on system 34. When the server program completes its task, the mirror program returns control to program 30 (PROG1) and passes any data back to system 32 via an allocated communication data area (COMMAREA).

The operation of the DPL is illustrated in rather more detail on Figure 4 in which a transaction 40 (AAAA) in system CICA includes a client program with a LINK command to a program PGA in a remote system CICB. In the client region the LINK command is passed over line 41 to the CICS command-level control programs 42. One of these, the command level EXEC interface program DFHEIP determines that the requested server program is on another system CICB. It calls a transformer program 43 (DFHXFP) over line 44 to transform the request into a form suitable for transmission. The EXEC interface program then calls an intercommunication component program DFHISP to send the transformer request over line 45 to the other system.

The intercommunication component uses CICS terminal-control facilities to send the request to the mirror transaction. The request to a particular server region causes the communication component in the client region to precede the formatted request with the identifier of the appropriate mirror transaction 46 to be attached in the server system.

This transaction name must be defined in the server region as a transaction that invokes the mirror program DFHMIRS (Fig. 3). To initiate any user-defined mirror transaction, the client program specifies the transaction name of the LINK request. Alternatively, the transaction name can be specified on the TRANSID option of the program resource definition.

As indicated by line 47, the mirror transaction uses an identical transformer program 48 (DFHXFP) to decode the formatted link request. The mirror then executes the corresponding command, thereby linking to the server program PGA. When the server program issues the RETURN command, the mirror transaction uses the transformer program to construct a formatted reply over line 49. Finally, over line 50, the mirror transaction returns this formatted reply to the client region. In that region (CICA) in the example), the reply is decoded, again using the transformer program, and used to complete the original request made by the client program. Further detail of the operation of the mirror transaction technique can be found in US Patent 4274139.

In order to explain how the invention is implemented using distributed program link, it is desirable first to discuss some fundamental aspects of transaction processing in CICS systems such as those of Figures 1 and 2. In particular, concepts of integrity and recoverability are important and depend on how transactions themselves are delimited.

Where many users have access to the same data, there is always a chance that several of them will try to change the same record at the same time. Rather than allow this to happen, CICS ensures that one operator's updating of data is complete before another's may start.

During normal execution of a task, CICS logs information about all protected data that is being changed. This protected data is called a recoverable resource. The term applies to any resource for which recovery information is recorded (and which can therefore be recovered by reversing or "backing out" the changes made to it). It covers, among other things, certain transient data and temporary storage queues, and databases.

CICS puts the information about this protected data in the dynamic log (an area of storage allocated to the transaction as required). The log information is deleted after the successful completion of the task.

However, if for any reason the task isn't completed, the data changes can be reversed, restoring the protected data to its original state.

CICS carries out dynamic transaction backout (DTB) to manage this recovery. If a transaction fails, due perhaps to application program error, data access error, transmission error, or because an operator decides to cancel a transaction, DTB reverses the updates that have been carried out by the transaction involved.

The process of cancelling changes in data works backwards from the last change before the failure, hence the name dynamic transaction backout. The backout occurs within the same task. This safeguards other tasks from the possibility of using corrupted data, because modified data is not released for use by them ("committed") until the current task has finished with it.

Application programs can specify intermediate synchronisation points (syncpoints). These are points at which data updates or modifications are logically complete. Sync points delimit a logical unit of work (LUW). If there is a failure, a sync point tells CICS that changes made before that point (that is, during a preceding LUW) do not need to be backed out. Sync points help to speed up and simplify recovery from failure in a long-running task.

If DTB is employed, it can be followed by a restart function which allows the transaction to be retried immediately.

Transaction restart is an optional facility that allows a cancelled transaction to be restarted automatically without intervention by the terminal operator, provided that certain conditions are met. The facility allows the application program to perform additional recovery functions written by the programmer.

If an application program runs into problems, it can choose to call the rollback facility to cancel the changes it has made in all

recoverable resources during the current transaction. These are then restored to the state prevailing at the previous syncpoint.

These transaction recovery functions are usually transparent to the terminal user.

CICS is a transaction manager that does not require the programmer to explicitly code START-TRANSACTION and END-TRANSACTION delimiters. Instead, CICS initiates programs within the scope of a transaction. One can imagine that CICS has issued a START-TRANSACTION request on behalf of the program before invoking it. After the program ends and returns control to CICS, CICS issues and END-TRANSACTION on behalf of the program.

CICS manages the execution units or threads (in CICS terminology these are called "tasks") under which the programs run. How the execution units are managed varies from one CICS implementation to another.

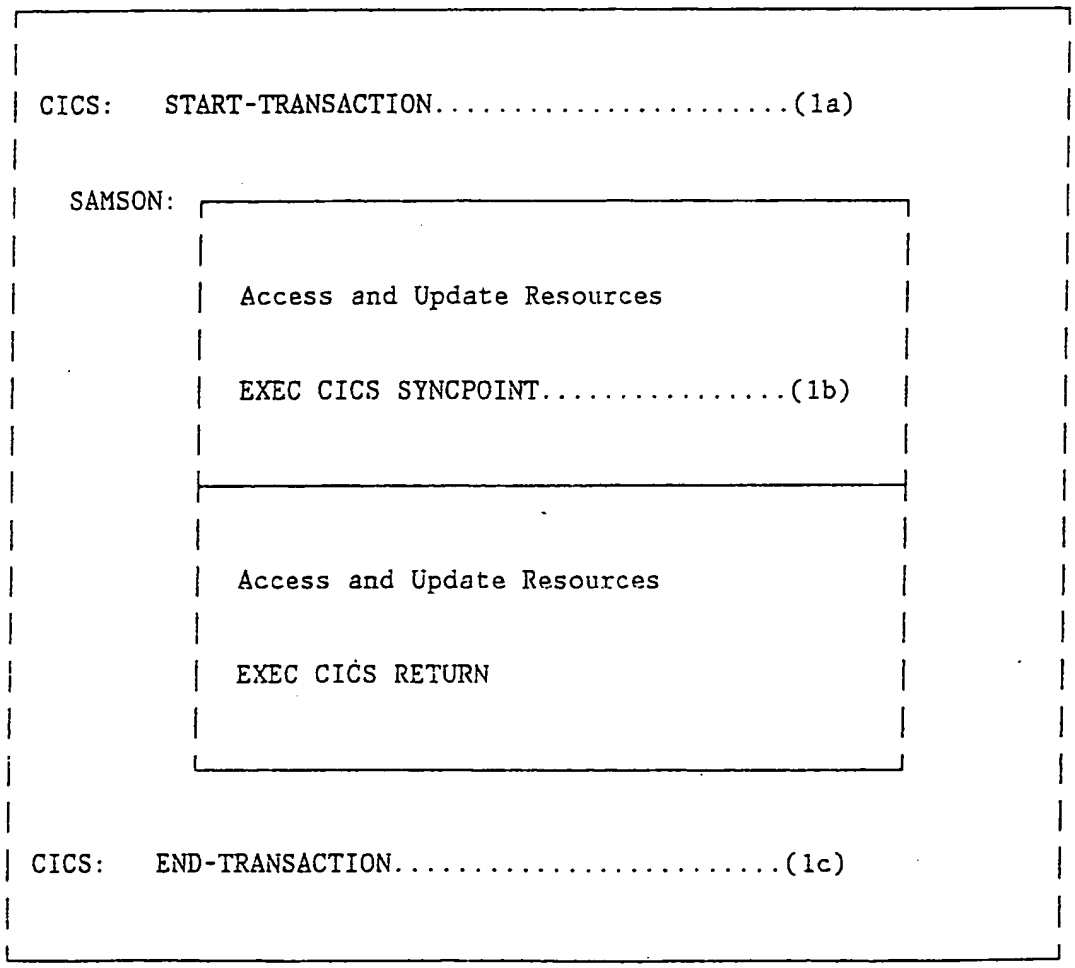
CICS also provides an application programming interface (API) for the programmer to split the transaction delimited by CICS into smaller consecutive transactions. This API is known as EXEC CICS SYNCPOINT, and can be thought of roughly as an END-TRANSACTION immediately followed by a START-TRANSACTION. Thus, it is impossible for the program to execute any code within the CICS environment without it being part of a transaction. At SYNCPOINT, all recoverable resources are committed and locks are freed. Cursor positions in files and databases are usually lost too, although in some cases changes to CICS and related products are being introduced to preserve cursors. Program and state storage is kept across SYNCPOINTs. The SYNCPOINT API also includes an option for the program to issue a ROLLBACK.

CICS also provides API for the program to return control to its caller which could either be another CICS program, or CICS itself. This is the EXEC CICS RETURN command.

A simple CICS transaction is delimited by the following:

- A CICS issued START-TRANSACTION or application issued SYNCPOINT
- A CICS issued END-TRANSACTION or application issued SYNCPOINT

Here is an example of a CICS program called SAMSON that has two transactions:



In this example there are two simple consecutive top-level transactions. The first is bounded by (1a) and (1b), the second is

bounded by (1b) and (1c). If the second transaction fails, the first transaction remains committed.

With long running programs, it is often necessary to use multiple transactions in this way to free locks and resources so that they can be accessed by other users. Nevertheless with a long running program, the execution unit continues to be used. For this reason it is advisable not to have many long running programs executing in a CICS system. In addition, not all resources are freed at SYNCPOINT.

A CICS program, SAMSON say, executes on behalf of a "user". That user could be a human being using a workstation or terminal, directly managed by CICS. In other cases, the user could be defined by another program, possibly non-CICS, that initiated SAMSON through CICS functions. The term "principal facility" is the CICS term for how the user is represented.

When a human user wants to run SAMSON, typically she/he would type in a four character code, e.g. HAIR. HAIR is associated with the SAMSON program in a CICS cross reference table, usually set up by a systems programmer. Thus, HAIR is the user's shorthand for SAMSON, but as we shall see later, it is also convenient to think of HAIR as a way of grouping execution time information, concerning SAMSON and related programs. This four character code is known in CICS as a "transaction code".

Both the user designation and the transaction code cannot alter once they have been allocated to an execution unit, until END-TRANSACTION is issued. CICS provides API (the ASSIGN command) for the program to access (but not update) the relevant context information.

The CICS program SAMSON can pass control to another CICS program, DELILAH say, in the same execution unit in two ways:

- EXEC CICS LINK to program DELILAH:

This is analogous to native programming language call, e.g. in PL/I, Call DELILAH; Native language call can be used instead, but the EXEC CICS LINK mechanism does provide additional functions.

- EXEC CICS XCTL to program DELILAH
  This means that SAMSON relinquishes control to DELILAH, and control never returns back to SAMSON.

With both these constructs, SAMSON can pass information to DELILAH in a data structure known as a COMMAREA (communications area). DELILAH can return information back to SAMSON in the same COMMAREA. In both cases, SAMSON and DELILAH are running in the same execution unit. Programs SAMSON and DELILAH execute as part of the same transaction. The transaction is terminated and a new one is started when either SAMSON or DELILAH issue an EXEC CICS SYNCPOINT. Programs SAMSON and DELILAH are both associated with the same transaction code HAIR. That is because the first program SAMSON in the execution unit was initiated by the user typing in the transaction code HAIR.

Returning now to the distributed programs link (DPL), this permits a CICS program, DELILAH say, to call another CICS program, SAMSON say, that is located in a different CICS system. In this case, DELILAH is referred to as the client program and SAMSON as the server program. This same API is used for DPL as for the regular EXEC CICS LINK. However, there are some optional extensions that can be used with DPL. A server program can act as a client with respect to another program. From the point of view of transaction types, there are two ways of using DPL, without and with an option on the LINK command known as SYNCONRETURN.

Without the SYNCONRETURN option, as illustrated in Figure 5, SAMSON runs as part of DELILAH's transaction. When DELILAH commits, all the recoverable resources updated by both SAMSON and DELILAH will commit. The server, SAMSON, is not permitted to commit. If either DELILAH or

SAMSON fail, then all the recoverable resources updated by both of them will be backed out.

In the example, DELILAH LINKs to SAMSON twice without the SYNCONRETURN option and then terminates, causing all resources updated by DELILAH and SAMSON to be committed. There is one transaction in this example.

In CICS terminology, the client commits the logical unit of work both for the client and server resources either with explicit commands in the client program or with an implicit syncpoint produced by CICS when the client task ends.

The example given, including only one server, is a simple one. In more complex configurations with multiple servers invoked, a full two-phase commit protocol is necessary to protect shared resources. In phase 1 of the process, the client as co-ordinator polls all servers to see if they are prepared to commit. Upon receiving a prepare-to-commit request, each server when ready, records the information necessary to redo or undo its part in the transaction in a dynamic log and responds with an agree-to-commit message.

The client enters phase 2 when it recoverably makes the decision to commit or abort the transaction. If all participants agree to commit, the client records the commit decision in the dynamic (redo) log and broadcasts the commit message to each participant. If any server does not agree to commit or respond within a time limit, the client records the abort decision in an undo log and broadcasts a restart message to each participant.

The servers respond by committing or backing out their part of the transaction as appropriate and acknowledging to the client. After the client has received all phase 2 acknowledgements, it either terminates the transaction or restarts it.

While two-phase commit is essential for the protection of shared resources accessed by distributed systems, the two-way flow of messages between the participants can take up a considerable time.

The SYNCONRETURN option on the distributed program link permits certain classes of distributed transaction to avoid the need for two-phase commit.

With the SYNCONRETURN option, as illustrated in Figure 6, SAMSON runs as a separate transaction to DELILAH. CICS starts a new top-level transaction before invoking SAMSON. When SAMSON completes, CICS ends SAMSON's transaction before returning control to DELILAH. SAMSON's transaction commits or backs out independently of DELILAH's.

In the example, Delilah LINKs to Samson and to Victor both with the SYNCONRETURN option and then terminates causing all resources updated by Delilah to be committed. Samson's resources are committed just before returning control to Delilah. The same applies to Victor. There are three transactions in this example, one for Delilah, one for Samson and one for Victor. When Delilah commits (CICS issues an implicit SYNCPOINT or End-txn when Delilah terminates), no co-ordination takes place with either Samson or Victor.

SYNCONRETURN specifies that the server region should take an independent SYNCPOINT on successful completion of the server program. The server's resources are committed in a separate logical unit of work immediately prior to returning control to the client, that is, an implicit syncpoint is issued for the server just before the server returns control to the client. SYNCONRETURN is intended for use when the client program is not updating any recoverable resources, for example, when performing screen handling. However, if the client does have recoverable resources, they are not committed at this point. They will be committed when the client itself syncpoints or in the implicit syncpoint at client task end.

The following restrictions apply to wrapped transactions, as implemented:-

- The wrapped transaction must run at a different location from outer transaction. There is an option to run it locally, but this is intended for testing purposes only.

- The outer transaction can initiate wrapped transactions serially only.

- The outer transaction is limited in the transaction context information it can redefine for the wrapped transaction. It can alter the transaction code, which is a grouping mechanism that has a major influence on CICS execution. The other pieces of context information are inherited by the wrapped transaction from the outer transaction.

It is possible to dynamically choose where the server is to run, e.g. for the purposes of work-load balancing, or to execute near the data it needs to access.

There follow some examples of applications where wrapped transactions are appropriate:

a) Updating remote central data

The outer transaction could be dealing with the end user interface portion of the application, and updating local recoverable data purely to gather the input from multiple user interactions, and to perform some validation. After a few user interactions, a single wrapped transaction could be used to update the remote central data. On completion of the remote update, the local data could be deleted. This is a typical client server example.

b) Long lived outer transaction

A long lived outer transaction could be processing sets of records, each set being associated with a different user. For each user encountered, a wrapped transaction could be initiated to deal with that user's records. An example of data of this type, is input (sequential files) arising from portable devices that are intermittently connected to the network. To ensure that the appropriate resource access security checks are made, it is usually vital that the wrapped transactions execute on behalf of the appropriate user (the real originator of the input), rather than a generic system user associated with the outer transaction. This is also important for chargeback purposes.

c) Read only wrapped transactions

Where a remote portion of a transaction is read only, and where it is not essential that the remote data be consistent with data at another location being updated within the same outer transaction, then a wrapped transaction is appropriate. Thus, there is no need to wait till the outer transaction commits to free any locks. A specific example could be where large amounts of less volatile data are held centrally, such as a customer file, and order data is held locally by department say. The departmental application that produces orders could access the central customer file to retrieve name and address information using a wrapped transaction, but elements and the wrapped transaction could execute within the scope of a single outer transaction.

d) Co-ordination of local departmental and remote central data

One example is where local and central data are replicated, and both are being kept up to date. Local data is used to satisfy all read requests. When an update is requested, a wrapped transaction modifies the central data. When that completes, the local data is updated. Alternatively, the local data could be updated first, but flagged that the central data has not yet been updated. The local

updates could become visible to other local users immediately. When the central updates are completed, the local data could be flagged to indicate the corresponding central updates are completed.

This approach can be used for multi-level updates and replication. By using "suspense" files and/or a general utility which intermittently checks for differences between the local and remote data, data consistency can be maintained.

CLAIMS

1. A method of processing transactions in a distributed transaction processing system comprising first and second transaction processing means capable of communicating with each other and with resources which may be used and updated in the processing of transactions, each of the transaction processing means being adapted to issue a commit message indicating, at the end of a respective transaction, that changes to said resources shall be committed; the method comprising the steps of:

initiating a transaction on said first transaction processing means, said transaction including a client program resident in said first transaction programming means which includes a program call to a related server program resident in said second transaction processing means;

passing said program call to said server program to cause said server program to run in said second transaction processing means;

in response to a first type of program call, permitting only the first transaction processing means to issue a commit message in accordance with a commit protocol so that both client and server programs run as part of a single transaction and any resource updates by either program are only committed in response to the commit message from the first transaction processing means; and

in response to a second type of program call, permitting the second transaction processing means to issue a commit message independently of the first transaction processing means so that the server program runs as part of a separate transaction and any resource updates by the server program are committed independently.

2. A method as claimed in Claim 1 in which resources are updated by the server program including the further steps of logging the updates as

they are made and restoring the resources to their original state if the second transaction processing means fails to issue a commit message in response to said second type of program call.

3. A method as claimed in Claim 1 or Claim 2 in which resources are updated by the server program including the further steps of locking the resources during updating and freeing them upon issue of a commit message by said second transaction processing means.

4. A method as claimed in any preceding claim including the step of detecting in the program call the absence or presence of an indication that independent commitment by the second transaction processing means is permitted, such absence or presence distinguishing the first and second types of program call respectively.

5. A method as claimed in any one of Claims 1 to 4 in which said commit protocol is a two-phase commit protocol.

6. A distributed transaction processing system comprising first and second transaction processing means capable of communicating with each other;

resources which may be used and updated by the processing means in the processing of transactions, each of the transaction processing means being adapted to issue a commit message indicating at the end of a respective transaction, that changes to said resources shall be committed;

the first transaction processing means including means for executing as part of a transaction a client program which includes calls to a related server program and the second transaction processing means including means for executing such a server program;

the system further including means for initiating a transaction including such a client program;

means for passing a program call from such a client program to such a related server program;

means responsive to a first type of program call to permit only the first transaction processing means to issue a commit message in accordance with a commit protocol so that both client and server programs run as part of a single transaction and any resource updates by either program are only committed in response to the commit message from the first transaction processing means; and

means responsive to a second type of program call to permit the second transaction processing means to issue a commit message independently of the first transaction processing means so that the server program runs as part of a separate transaction and any resource updates by the server program are committed independently.

7. A system as claimed in Claim 6 in which the first and second transaction processing means are physically separate and independent systems and are in communication via a communication link.

8. A system as claimed in either Claim 6 or Claim 7 including log means for logging updates to the resources by such a server program as they are made and restoring means for restoring the resources to their original state if the second transaction processing means fails to issue a commit message in response to said second type of program call.

9. A system as claimed in any one of Claims 6, 7 or 8 including resource locking means for locking resources during updates by the server program, the locking means being responsive to the issue of a commit message by the second transaction processing means to free the locked resources.

10. A system as claimed in any one of Claims 6 to 9 including means for detecting in a program call, the absence or presence of an indication

that independent commitment by the second transaction processing means is permitted.

11. A system as claimed in any one of Claims 6 to 10 in which said commit protocol is a two-phase commit protocol.